**EE260 Lab 2 Sign-Off Sheet**

Make sure lab instructor/TAs initial and date each part. Attach this sheet and the Report Grading Rubric to your team's lab report!

Your Name:

Lab Partner: Date Performed:

## Demonstrated correctly:

•Pre-lab Complete (1) (2) (10 pts, individually)

* + Verilog Full Adder (15pts)
  + Verilog 4-bit Adder (20 pts)
  + TA Questions: (1) (2) (5 pts, individually )



**EE260: Introduction to Digital Design**

**Lab 2 – Implementing A Full Adder**

# Objective:

In this lab exercise, you will simulate, test, and download various digital circuits which implement some common combinational functions as an adder. You will again simulate and implement your circuits completely using Verilog code.

# Pre-lab Assignment:

This pre-lab assignment is ***to be completed by each team before your lab session*** and must be signed-off by the TA during your lab session. Pre-labs help you to become oriented to the problem before you enter the lab, help complete your design in advance and prevent wasting time in the lab. Include the signed pre-labs for both team members in your report.

1. READ the whole lab assignment!
2. A full adder, as we discussed in class, is a circuit that adds 2 bits *A* and *B* plus any possible *Carry* from a less significant bit. For the full adder block diagram below:
   1. Refer to the posted notes from Lecture 4 for the truth table for a Full Adder. Write out the full Sum of Products expressions for *Cout* and *Sum.*

|  |  |  |
| --- | --- | --- |
| *Cin* | Full | *Cout* |
| *A B* | Adder | *Sum* |

*Figure 1*

1. Using the block diagram of the 1-bit full adder shown above, draw a schematic showing how you would implement a 4-bit adder which could add any two 4-bit binary numbers like

A3 A2 A1 A0

+ B3 B2 B1 B0

S3 S2 S1 S0

using four full adders. Be sure to label all inputs and outputs and show how the carry out of each less significant bit becomes the carry in of the next more significant bit (but carry-in of the LSB is always 0).

1. Write out a Boolean expression for detecting OVERFLOW when adding two 4-bit

*signed* numbers.

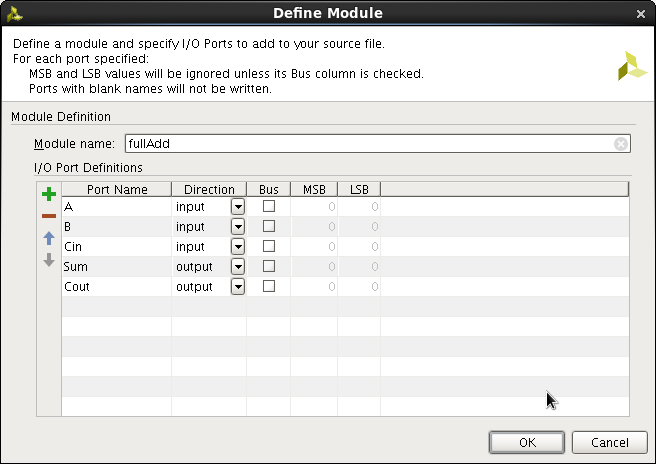
OF\_S =

1. Write out a Boolean expression for detecting OVERFLOW when adding two 4-bit

*unsigned* numbers.

OF\_U =

# Lab Assignment:

In this lab, you will implement your Full Adder design, and then you will use your 1-bit adder to implement a 4-bit adder. Refer back to the Lab 1 tutorial for detail on creating new projects, adding sources, performing simulations, etc.

1. Open Vivado and create a new project correctly setting the target FPGA properties as you did in Lab 1. Add a new Verilog source module and call it *fullAdd.v*.
2. Fill out the Port-assignment Wizard according to the figure below with inputs *A, B,* and *Cin* and outputs S*um* and *Cout*. Remember that Verilog, like C, is case sensitive and *A* is considered a different variable from *a*. Vivado will generate the Verilog module shell with those inputs and outputs declared. REMEMBER TO FILL IN THE COMMENTS BLOCK with your name(s) and a brief description of the module!!

You will need to add the assign statements to implement your full-adder expressions for

*Cout* and *Sum*. In Verilog, the Boolean operations are written as

**&** for AND, **|** for OR, **~** for NOT and **^** for XOR.

1. Unlike most calculators and many software programming languages, ***there is NO implied order of operations among the BOOLEAN operators in Verilog****.* ***You must enforce an order of operations with*** ( ).

Ex: F = (A B' + C)' + A'C

// Verilog assignment for F

assign F = (~((A & (~B)) | C)) | ((~A) & C);

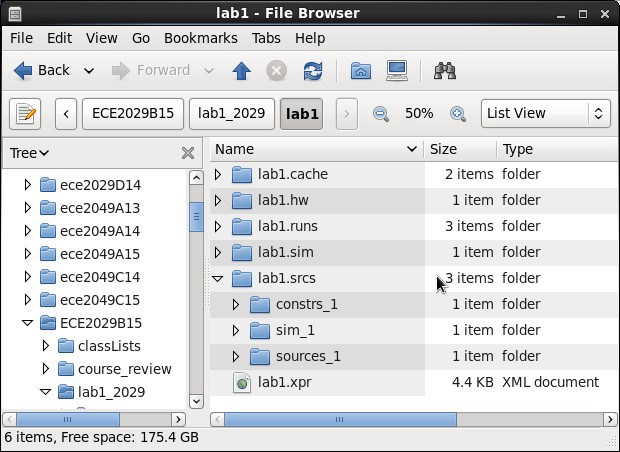
Implement the Verilog assignment statements for the expressions you generated for *Cout*

and *Sum* in the module and save.

1. Drawing on the constraints file from Lab 1 as an example, add constraints (.xdc) file to your project that assigns inputs *A*, *B* and *Cin* to switches SW0, SW1 and SW2 respectively and assigns outputs *Sum* and *Cout* to LEDs LD0 and LD1 respectively. The constraints file is stored in a directory called something like

"proj\_name/proj\_name.srcs/constrs\_1".

Below is this directory structure for my lab one project. Note that I run Linux, but you will see a similar directory structure in Windows Explorer for your lab project directory.

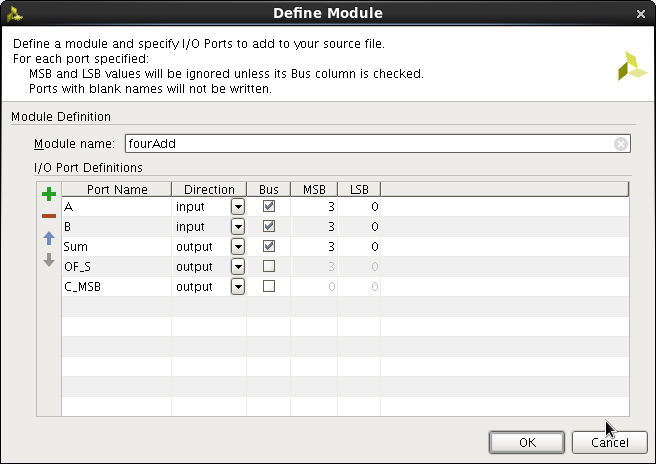


1. Run Synthesis. You should see no errors.
2. Add a source for the simulation to implement a full-adder test bench (as you did in Lab 1). Add Verilog to instantiate your fullAdd module as U1 and apply test inputs for all possible input combinations of *A*, *B,* and *Cin*. Remember that you will need to make register declarations to define the input variables to your U1 module and wire declarations for the output variables. This is because the simulation is run entirely within the development environment. It does not use the input/output constraints associated with the fullAdd.v source Run a Behavioral Simulation to verify that you circuits are behaving correctly. ***Capture these ISim results for your lab report*.**
3. Run Implementation and then run a Post Implementation Timing Simulation. Measure the timing delay for *Cout* for when A=1 B=0 transitions to A=1 B=1. ***Capture these ISim results for your lab report*.**

1. Generate Bit file and Program Device. Verify that your full adder is working properly then ***demonstrate it to the TA for sign-off***.

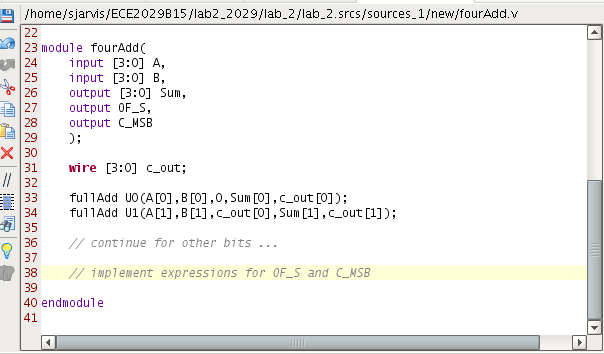
# Implementing a 4-Bit Adder in Verilog

1. Save your project then add a new Verilog Design Source, called *fourAdd.v,* and *fill in it's comment block*. In this case your inputs and outputs will be 4-bit buses. See the example Port Declaration below. Notice that this module has no inputs for carries. That is because carry from one column to the next is part of the internal workings of this circuit. The user does not need (or want) to see that. Rather the user expects to input two 4-bit numbers and get the correct 4-bit result (or be flagged that an error has occurred).



1. Right click on *fourAdd.v* then Set as Top. In this Verilog module, you will “instantiate” copies of the *fullAdd* module. To instantiate a module, you need to call it something unique so that Verilog can separate the fullAdd used for bit 0 from the one used for bit 3.

Give each of the four *fullAdd*-ers you instantiate a U#. See the code example below. You will also need internal connections (called “wires”) to convey carry information from one fullAdd module to the next. These wires are not inputs or outputs, so they weren't defined when we did the port declaration. They are like labeled intermediate points on a circuit diagram.



1. Use assign statements to implement the signed overflow flag, OF\_S, and the unsigned overflow flag, which is the carry-out of the most significant bit, C\_MSB.
2. Right click on your *fullAdd.xdc* constraint file and remove it from the project (you should not delete it, however). There can only be one constraint file active in a project at a time. Now add a new constraints file (*fourAdd.xdc*) to your project and assign inputs *A[3:0]* to *SW3-SW0* (shown below) and *B[3:0]* to SW7-SW4. Also assign outputs *Sum[3:0]* to LEDs LD3-LD0 and OF\_S to LD15 and C\_MSB to LD14 respectively. Remember, pin assignments are given in Figure 16 of the [Basys 3 Reference Manual](https://www.digilentinc.com/Data/Products/BASYS3/Basys3_rm.pdf). Run Synthesis.

#Switches

**set\_property** PACKAGE\_PIN V17 [**get\_ports** {A[0]}]

**set\_property** IOSTANDARD LVCMOS33 [**get\_ports** {A[0]}]

**set\_property** PACKAGE\_PIN V16 [**get\_ports** {A[1]}]

**set\_property** IOSTANDARD LVCMOS33 [**get\_ports** {A[1]}]

**set\_property** PACKAGE\_PIN W16 [**get\_ports** {A[2]}]

**set\_property** IOSTANDARD LVCMOS33 [**get\_ports** {A[2]}]

**set\_property** PACKAGE\_PIN W17 [**get\_ports** {A[3]}]

**set\_property** IOSTANDARD LVCMOS33 [**get\_ports** {A[3]}]

# .... ADD other switches and LEDs

1. Add a new source for simulation and create a test bench for your *fourAdd* module. Make sure to set your new TB as the top. (If there is a problem, you may need to remove your fullAdd testbench. Again, don't delete it!) There are 256 possible input combinations

for A and B. Select a handful of inputs to test in your simulation. Be sure to set one or two that will activate OF\_S and/or C\_MSB. ***Run a Behavioral Simulation and save the ISim output for your report.***

1. Run Implementation and a Post Implementation Timing Simulation. Measure the longest timing delay for the input cases you chose. ***Capture these ISim results for your lab report*.**
2. Generate Bit file and Program Device. Verify that your 4-bit adder is working properly then ***demonstrate it to the TA for sign-off***.
3. Write a high-quality Lab Report which includes all your Verilog code, constraint, files, testbenches, simulation results, your pre-labs, and the grading rubric.

**ALL TEAM MEMBERS SHOULD SAVE THIS PROJECT TO YOUR USB DRIVES RIGHT NOW!!!**